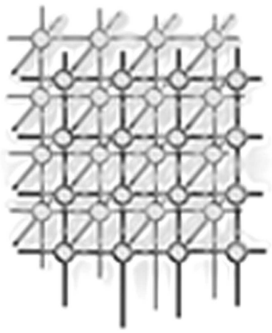
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 00; **00**:1–7



Coordinated Rescheduling of Bag-of-Tasks for Executions on Multiple Resource Providers

Marco A. S. Netto*,†, Rajkumar Buyya

Cloud Computing and Distributed Systems (CLOUDS) Laboratory
Dept. of Computer Science and Software Engineering
The University of Melbourne, Parkville, VIC 3010, Australia

SUMMARY

Metaschedulers can distribute parts of a Bag-of-Tasks (BoT) application among various resource providers in order to speed up its execution. The expected completion time of the user application is then calculated based on the run time estimates of all applications running and waiting for resources. However, due to inaccurate run time estimates, initial schedules are not those that provide users with the earliest completion time. These estimates increase the time distance between the first and last tasks of a BoT application, which increases average user response time, especially in multi-provider environments. This paper proposes a coordinated rescheduling algorithm to handle inaccurate run time estimates when executing BoT applications in multi-provider environments. The coordinated rescheduling defines which tasks can have start time updated based on the expected completion time of the entire BoT application. We have also evaluated the impact of system-generated run time estimates to schedule BoT applications on multiple providers. We performed experiments using simulations and a real distributed platform, Grid'5000. From our experiments, we obtained reductions of up to 5% and 10% for response time and slowdown metrics respectively by using coordinated rescheduling over a traditional rescheduling solution. Moreover, coordinated rescheduling requires little modification of existing scheduling systems. System-generated predictions, on the other hand, are more complex to be deployed and may not reduce response times as much as coordinated rescheduling.

KEY WORDS: Rescheduling; bag-of-tasks; resource allocation; grid computing; metascheduling; parallel computing; performance prediction; run time estimates; quality-of-service

*Correspondence to: Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Dept. of Computer Science and Software Engineering, The University of Melbourne, Parkville, VIC 3010, Australia
†E-mail: marco.netto@gmail.com

Received 9 July, 2010
Revised

1. INTRODUCTION

Bag-of-Tasks (BoTs) are parallel applications with no inter-task communication. A variety of problems in several fields, including computational biology [1], image processing [2], and massive searches [3], have been modeled as BoT applications. In comparison to the message passing model, BoT applications can be easily executed on multiple resource providers to meet a user deadline or reduce the user response time. Although BoT applications comprise independent tasks, the results produced by all tasks constitute the solution of a single problem. In most cases, users need the entire set of tasks completed to be able to post-process or analyse the results. Therefore, the optimization of the aggregate set of results is important, and not the optimization of a particular task or group of tasks [4, 5]. Some examples of BoT applications are a set of frames to render a movie or a set of biological experiments to understand how a molecule works.

Metaschedulers can distribute parts of a Bag-of-Tasks (BoT) application among various resource providers in order to speed up its execution. The expected completion time of the user application is then calculated based on the run time estimates of all applications running and waiting for resources. A common practice is to overestimate execution times in order to avoid user applications to be aborted [6, 7]. Therefore, initial completion time promises are usually not accurate. In addition, when a BoT application is executed across multiple providers, inaccurate estimates increase the time difference between the start of the first task and the completion of the last task of a BoT, which increases average user response time. This time difference, which we call *stretch factor*, increases mainly because rescheduling is performed independently by each provider.

System-generated predictions can reduce inaccurate run time estimates and avoid users having to specify these values. Several techniques have been proposed to predict application run times and queue wait times. One common approach is to analyze scheduling traces; i.e. historical data [8, 9, 10]. Techniques based on trace analyses have the benefit of being application independent, however may have limitations when workloads are highly heterogeneous. Application profiling has also been vastly studied to predict execution times [11, 12, 13, 14]. Application profiling can generate run time predictions for multiple environments, but usually requires application source code access. Therefore, even though techniques for generating predictions are available, they have limitations. One alternative solution for reducing the stretch factor generated mainly by the inaccurate run time estimates, and hence reduce user response time is through proper rescheduling of BoT tasks.

This paper proposes a coordinated rescheduling strategy for BoT applications running across multiple resource providers. Rather than providers performing independent rescheduling of tasks of a BoT application, the metascheduler tracks the expected completion time of the last BoT task on all involved providers. This strategy minimizes the stretch factor and reduces user response time, which is particularly important when it is not possible to obtain accurate run time estimates of user applications. We also show that on-line system generated predictions, even though require time to be obtained, can reduce user response time when compared to user estimations. Moreover, with more accurate predictions, providers can offer

tighter expected completion times, thus increasing system utilization by attracting more users. We performed experiments comparing the benefits of using coordinated rescheduling over uncoordinated rescheduling using both simulations and a real distributed platform, Grid'5000, on homogeneous and heterogeneous resources. We obtained very positive results on user response time that can be achieved with minor modification of a typical metascheduling system.

2. RELATED WORK

Extensive research has been done on the areas of performance prediction techniques [15, 12, 11, 13, 8, 9, 10, 16, 17], use of predictions for scheduling applications [18, 19, 17, 20], and scheduling algorithms for BoT applications [21, 22, 23]. In this section, we present a few projects on use of predictions for scheduling applications and scheduling of BoT applications, which are the two main areas related to our work.

In order to use predictions for scheduling applications, scheduling systems can mostly rely on techniques that use historical data of previous application executions [8, 9, 10, 16, 17]. For instance, Tsafrir et al. [8] proposed the use of average time of the previous two executions of jobs with same characteristics to calculate new estimation time to be used for backfilling, whereas Smith et al. [10] use search for similar jobs of the entire historical data base of previous executions. Run time predictions also assist scheduling systems to provide users with expected waiting times [10] and to produce schedules that increase system utilization [16].

He et al. [18] addressed the problem of dynamic scheduling for parallel jobs in multi-cluster systems using performance predictions. Their work considers applications with deadline constraints and the performance predictions assist the scheduler to make decisions on work distribution. Berman et al. [19] have also proposed the use predictions for scheduling applications in the context of the AppLeS project. Tasks from "AppLeS-enabled applications" are rescheduled during execution by analyzing predictions of availability and characteristics of processors and network resources. Sonmez et al. [17] studied the use of time series prediction methods for job run times and queue wait times in Grid environments. They have also evaluated the impact of predictions when scheduling jobs in Grids. Their experiments consider schedulers using FIFO without backfilling and tasks of a BoT application are submitted to a single cluster.

Existing work on BoT applications mostly focuses on the initial scheduling [24, 21, 25, 23, 26, 22]; tasks are scheduled without considering the dynamic behavior of resources and applications. Researchers have also developed task replication techniques to reduce user response time and handle the lack of information from resources and tasks [27]. Task replication is a particular type of rescheduling, but with the drawback of wasting resources. Therefore, our contribution is a coordinated rescheduling algorithm for BoT applications and an evaluation of impact of run time estimates when scheduling these applications across multiple providers.

3. SCHEDULING ARCHITECTURE

A metascheduler receives user requests to schedule BoTs on multiple autonomous resource providers in on-line mode. Providers have no knowledge about one another. Users provide the

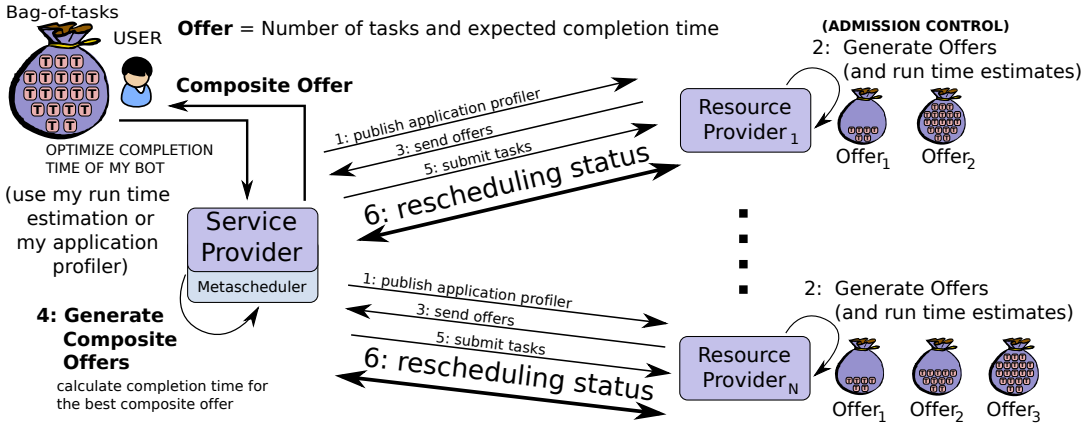


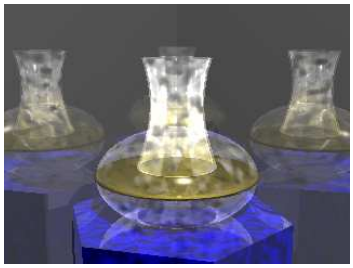
Figure 1. Components' interaction to schedule a Bag-of-Tasks on multiple resource providers.

number of required processors along with either a run time estimation or an application profiler. The application profiler is used by the scheduling architecture to automatically determine run time estimations inside each provider. The run time estimation can be of either the entire BoT or the tasks individually. The processors are available from space-shared machines such as clusters and massively parallel processing machines. We define a BoT as job composed of tasks. However, when a set of tasks from a BoT are submitted to a provider, we call this set as a job as well; thus BoTs also comprise jobs that run on multiple providers.

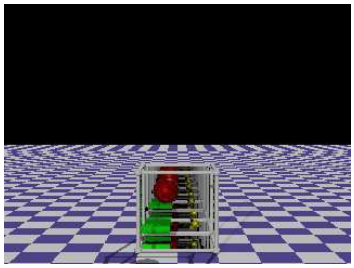
As illustrated in Figure 1, the scheduling of a BoT application consists of 6 steps. In step 1, the metascheduler exposes the application profiler or user estimations to the resource providers. In step 2, resource providers execute the profiler (if available) and generate a list of offers that can serve the entire BoT or only part of it. An offer consists of a number of tasks, and their expected completion time. In this work, resource providers generate offers that do not violate the expected completion time of already scheduled jobs. Once the resource providers generate the offers, they send them to the metascheduler (step 3), which composes them according to user requirements (step 4), and submits the tasks to resource providers (step 5). After the tasks of a BoT are scheduled, resource providers contact the metascheduler whenever tasks need to be rescheduled (step 6).

Due to the heterogeneity of processors, network, and size of scheduling queues in each resource provider, offers may reach the metascheduler at different times. Once the metascheduler receives all offers, some of them may no longer be valid since other users submitted applications to the providers. To overcome this problem, we use a similar approach developed by Haji et al. [28], who introduced a Three-Phase commit protocol for SNAP-based brokers. Our protocol uses *probes*, which are signals sent from the providers to the metaschedulers interested in the same processors to be aware of processor status' changes.

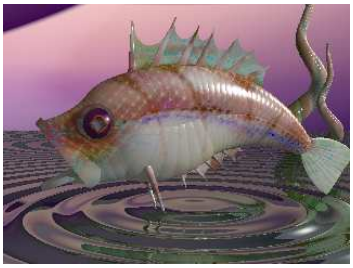
The **schedulers' goal** is to provide users with expected completion time and reduce such a time as much as possible during rescheduling phases.



(a) Sky Vase.

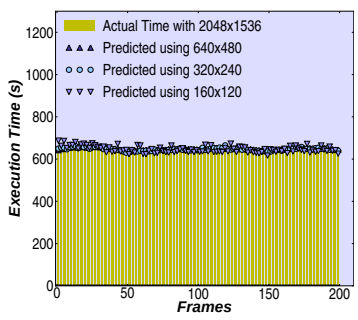


(b) Box.

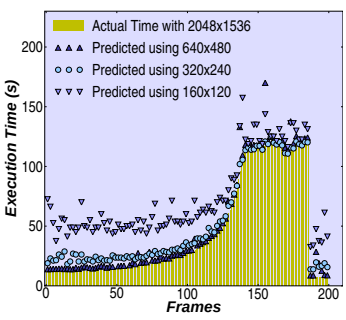


(c) Fish.

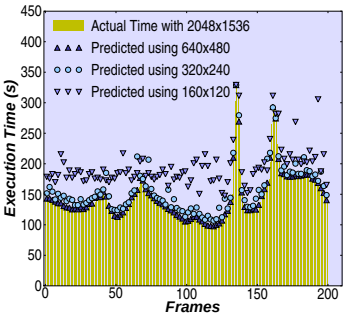
Figure 2. Example of images for each of the three animations.



(a) Sky Vase.



(b) Box.



(c) Fish.

Figure 3. Predicted execution time using most CPU consuming frame as base for estimations.

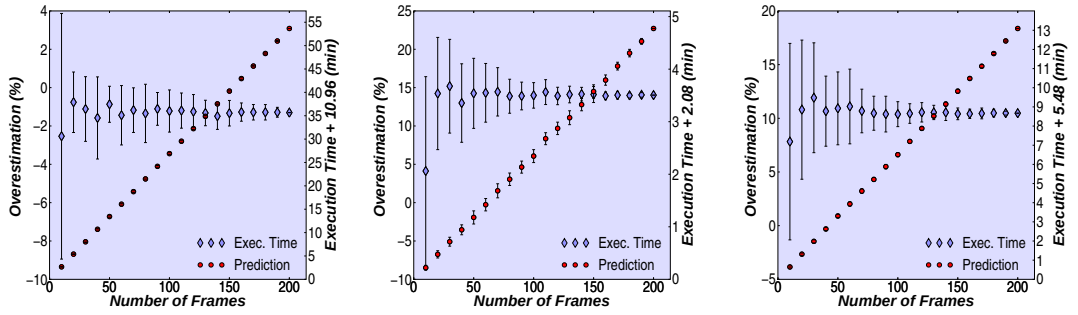
The next section presents an example of an application profiler that can be incorporated into the scheduling architecture.

3.1. Example of Application Profiler

In environments where resource providers work with precise run time estimates, metaschedulers can better distribute the tasks, thus reducing applications' response time, and resource providers can publish offers with tighter response times, thus increasing system utilization by attracting more users. This happens because the metascheduler knows the exact load in each provider. One approach to generate run time estimations is through application profiling via sampling execution. As prediction techniques are application dependent, this section just exemplifies the feasibility of obtaining run time predictions, since techniques to obtain them is out of the scope of this paper.

Table I. Time to generate execution time estimates.

Animation	Resolution	Exec. Time (min)	Perc. of total time	Accuracy
Sky Vase	640x480	223	10.3	0.1% underest.
	320x240	64	2.9	1.3% underest.
	160x120	27	1.2	0.2% underest.
Box	640x480	18.4	12.3	7.30% overest.
	320x240	6.8	4.5	14.00% overest.
	160x120	4.0	2.6	64.90% overest.
Fish	640x480	53.1	11.0	3.03% overest.
	320x240	18.57	3.9	10.47% overest.
	160x120	9.90	2.0	37.64% overest.



(a) Sky Vase. (b) Box. (c) Fish.

Figure 4. Predicted execution time using partial sampling of 320x240 frames.

POV-Ray[†] is a ray-tracer tool for generation of three dimensional images—a set of images can thus be used to create animations. We used POV-Ray to create three short animations containing 200 frames each, with a resolution of 2048x1536 pixels (Quad eXtended Graphics Array). The animations are based on image specifications that come with the POV-Ray package, namely Sky Vase, Box, and Fish. Sky Vase consists of a Vase with a sky texture on a table with two mirrors next to it. To generate the animation for Sky Vase, we included 360-degree rotation of the vase. Box consists of a chess floor with a box containing a few objects with mirrors inside. We therefore included a camera that gets closer to the box and crosses it on the other side. Fish consists of a fish over water that rotates 360 degrees. Different from Sky Vase, Fish has a more heterogeneous animation due to the shape of the fish. An example of images for each of the three animations is illustrated in Figure 2.

[†]POV-Ray project: http://www.povray.org

One technique to obtain the run time estimations is to render the animation in a much lower resolution and render a *base frame* in the actual resolution. Using the execution time of the base frame of the actual and the reduced animation, it is possible to generate a factor to be multiplied on the lower resolution animation to predict the execution actual time of each frame. Figure 3 presents predictions using the maximum and execution time frame as base frame. For this experiment, we used 640x480, 320x240, and 160x120 as lower resolutions for generating predictions. We also observe that both 640x480 and 320x240 resolutions provided much better predictions than 160x120. For the Box animation using the base frame with minimum execution time, all resolutions provided inaccurate predictions for long execution time frames. This happens because the base frame is too small to capture the differences between the resolutions. Table I summarizes the execution times to generate the predictions and their accuracies using the base frame with maximum execution time. The results show that good predictions are time consuming since we are using the entire animation.

It is also possible to reduce the profiling time by sampling a set of frames with a lower resolution rather than using the entire animation. Figure 4 shows the execution time and the prediction accuracy as a function of the number of frames sampled using resolution 320x240, using the maximum execution time as base frame. This technique is feasible because in an animation, neighbor frames have similar content and, depending on the case, the variation is minimum during the entire animation, such as for Sky Vase. Note that for some cases, there are negative overestimations, i.e. the execution times are underestimated. When allocating resources in clusters, a common practice is to kill applications when the actual execution time exceeds estimation execution time. Therefore, rescheduling happens only for overestimations; and for this reason, from now on, in this paper we use the term inaccurate estimation for overestimations only.

3.2. Where to Generate the Estimations

Either users or resource providers can generate run time estimates. If users know the exact resource configuration for each provider, users can execute the application profiler to obtain the run time estimations. Also, even if users do not know the configuration, providers that work with virtual machines can make them available to users. In this case, the users can run the generator through virtual machines in their local machines. Another option is to allow providers to generate run time estimates at the moment users submit their application requirements to the metascheduler. In this case, providers can have a dedicated set of processors to generate run time estimates, or providers can generate them by placing estimator jobs in their shared processors for actual executions. All these options depend on the application, environment settings, and user needs.

4. COORDINATED RESCHEDULING

Once the metascheduler provides the user with an expected completion time of a BoT application, tasks can start execution either immediately or after processors become available.

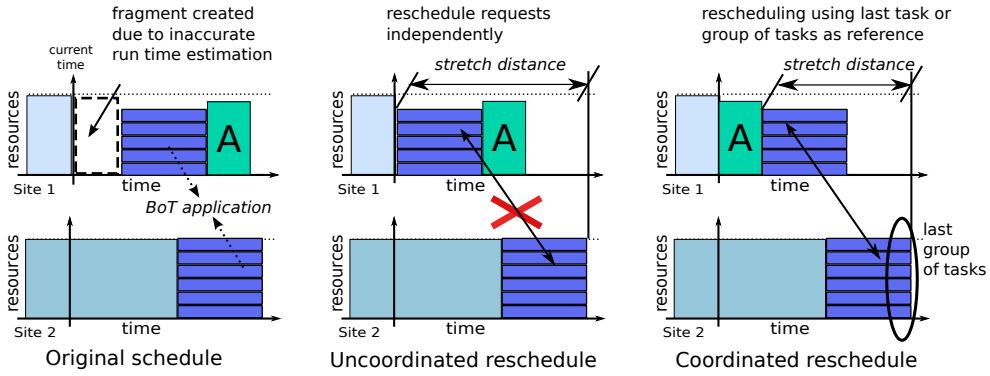


Figure 5. Example of schedule using coordinated rescheduling.

For the second case, it means tasks are placed in a waiting queue and can be rescheduled to start before expected when tasks from other applications have inaccurate run time estimates.

Figure 5 illustrates the difference between the traditional rescheduling strategy, which considers all tasks independently, and the coordinated rescheduling, which considers the tasks of a BoT as being part of a single application. In this example, by using uncoordinated rescheduling the BoT application tasks in Site 1 are rescheduled without considering the tasks from Site 2; which increases stretch factor of this BoT, and delays the completion time of application A. By using the coordinated rescheduling approach, application A is rescheduled first since its earlier start time does not increase the *overall* completion time of the BoT application. A formal definition of the stretch factor of a BoT application k composed of n tasks is:

$$SFactor(BoT_k) = n \left(\frac{T_n^c - T_1^s}{\sum_{i=1}^{n} T_i^e} \right)$$

Where, T_n^c is the completion time of the last task, T_1^s is the initial time of the first task, and T_i^e is the execution time of task i of a BoT with n tasks.

4.1. Rescheduling Algorithm

Whenever a job completes before the expected time, local schedulers execute Algorithm 1 to reschedule the waiting queue. The first step is to sort the jobs in the waiting queue by increasing order of their expected completion times (Line 1). Tasks from the same BoT are sorted by increasing order of expected start time individually. Later, tasks (or jobs, i.e. groups of tasks) are rescheduled one by one (Lines 2-9). For each job j_i being part of a BoT, the scheduler verifies whether j_i holds the expected completion time of the entire BoT. Both BoT jobs and other type of jobs are then rescheduled using FIFO with conservative backfilling

Algorithm 1: Pseudo-code for rescheduling jobs, which is executed on the local schedulers when a job completes before the expected time.

```
1  Sort jobs by expected completion time. BoT tasks are sorted by expected completion of
   the entire BoT. Tasks from the same BoT are sorted by expected start time
2  for ∀j_i ∈ waiting queue do
3      isLastTask ← false
4      previousCompletionTime ← j_i's completion time
5      if j_i is part of a BoT then
6          isLastTask ← holds the last expected completion time
7      Reschedule j_i using FIFO with conservative backfilling
8      if previousCompletionTime ≠ newCompletionTime and isLastTask = true then
9          add task to possible new completion time list
10 Check new completion times
11 Send new completion times
```

(Line 7). If a BoT job holds the expected completion time of the entire BoT and received a new completion time due to rescheduling, the algorithm keeps this job in a structure called *newCompletionTimes*, which contains the job id and the new completion time (Line 8-9). After all jobs are rescheduled, the algorithm analyses the last completion time for each BoT in the *newCompletionTimes* structure (Line 10). The local scheduler sends this structure to the metaschedulers holding the respective BoTs (Line 11).

From the metascheduler side, each time it receives the *newCompletionTimes* structure, it verifies whether the new completion times are local or global. If they are global (i.e. for the entire BoT), the metascheduler sends the new global completion time to the resource providers holding the respective BoT tasks.

4.2. Implementation

Figure 6 represents a simplified version of the class diagram for the metascheduler. There are four main components: the metascheduler main class, scheduler, rescheduler, and a list of scheduled jobs. The main responsibilities of the metascheduler class are to submit jobs to resource providers, and keep a table with the expected completion time of the BoTs. The complexity of implementing the scheduler lies on composing the offers. The rescheduler is responsible for updating BoT completion times in the scheduling queues of resource providers.

Existing local schedulers require an extension of the data structure that keeps the jobs in the system. This extension is the global completion time of the entire BoT that may have tasks managed by local schedulers from other resource providers. Local schedulers also contact the metascheduler to inform about new completion times. Therefore, minor changes are required to implement the coordinated rescheduling technique into existing scheduling systems.

5. EVALUATION

This section evaluates the coordinated rescheduling algorithm and the impact of inaccurate run time estimates when scheduling BoT applications on multiple resource providers. We performed experiments using both a simulator and a real testbed. Simulations have allowed us to perform repeatable and controllable experiments using various parameters. The experiments in a real testbed allow us to verify how the scheduler architecture can be used in practice. We have used our event-driven simulator, named PaJFit (Parallel Job Fit), and workloads produced by the Lublin-Feitelson model [29]. For the real experiments, we used an extended version of PaJFit [5], which works with sockets for communication between modules on Grid'5000. Following we describe the experiment configuration and the analysis of the results.

5.1. Experimental Configuration

We have set up a computing environment with a metascheduler and four clusters, C_{1-4}, with 300 processors each. From this environment, we have explored a set of scenarios as described in Table II. The set of experiments with all clusters with the same configuration helps us to analyze the differences between system and user generated estimations and the rescheduling algorithm. The experiment with all clusters with the same configuration but using different policies for run time estimates helps us to understand which resource provider would benefit more from each approach. We also analyze the impact of heterogeneity for scheduling and rescheduling of bag-of-tasks across multiple providers. Note that although there are larger systems available nowadays, by increasing the number of cores in the systems, the tasks to be processed would require to be more complex to explore those cores, for instance, by using multiple threads. Therefore, for this reason we adopted this scale, i.e. to make compatible the computing environment and the workload used for the experiments.

Availability and research on real workloads and workload models for BoT applications in Grid environments are in their early stages [23]. Several recent Grid studies utilize workload models using various statistical distributions to evaluate scheduling policies [30, 31]. We therefore preferred to rely on a well-established workload model for cluster-based parallel applications, proposed by Lublin and Feitelson [29], to generate traces for both the simulations and the experiments in Grid'5000. Note that the model is for parallel applications, and Bag-of-Tasks are a common type of parallel application highly used in the scientific community. Therefore, the workloads used in the paper represent any type of application that can run in

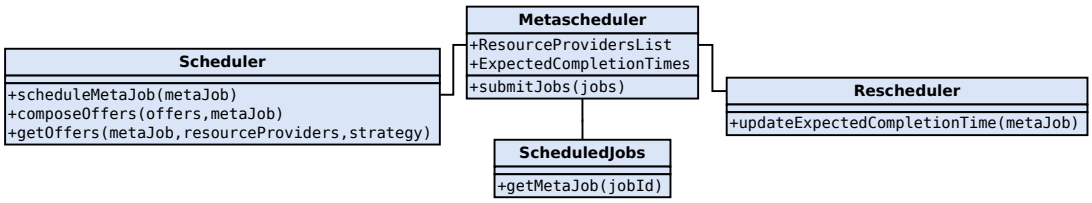


Figure 6. Class diagram for the metascheduler of BoT applications.

Table II. Main scenarios for the experiments.

Hardware	**Estimation Type and rescheduling**
C_1=C_2=C_3=C_4	UE with independent rescheduling
C_1=C_2=C_3=C_4	SE 50% and 80% more accurate than UEs with uncoordinated rescheduling and 5-30min generation time
C_1=C_2=C_3=C_4	UE with coordinated rescheduling
C_1=C_2=C_3=C_4	C_1 and C_2 with UEs and C_3 and C_4 with SEs
C_1=C_2 20 % and 50% faster than C_3=C_4	UEs
C_1=C_2 20 % and 50% faster than C_3=C_4	SEs
C_1=C_2 20 % and 50% faster than C_3=C_4	UEs with coordinated rescheduling

those requests, which can be Bag-of-tasks or not. In addition, we found reliable to use the workload model as it represents workloads that have been highly used in literature, especially to develop scheduling policies [8].

For each job produced by the model, we considered it as a BoT application, and its number of requested processors is the number of tasks in the BoT. From the model, we also used the submission and execution times. We simulated 15 days of the workload and used 10 workloads for each experiment in order to have a comprehensive set of data points. We also considered 10 overestimation percentages; from 0% to 200%, where 0% means the estimation is the exact execution time. We chose 200% as a limit because after this value, the chances of rescheduling becomes minimal in our experiments. Therefore, for each scenario described in Table II, we have a total of 100 simulations. For all experiments we set up the system load as 70% by changing the job arrival times due to the difference of the original load for each workload generated by the model. To achieve this load we used a strategy similar to that described by Shmueli and Feitelson to evaluate their backfilling strategy [32], but we fixed the time interval and brought more jobs from the same traces that would be out of the interval.

We performed our experiments in Grid'5000 by placing a local scheduler in four clusters with access to 300 processors. Figure 7 illustrates the location of the resource providers. Table III presents an overview of the node configurations in which we deployed the local schedulers and the metascheduler.[‡]

5.2. Results and Analysis

There are two factors related to the reduction of user response times: work distribution and backfilling. Work distribution can be improved by having better run time estimates, since the metascheduler can decide the right amount of work to send to each provider, whereas backfilling can fill queue fragments generated by earlier completion times of user requests. These fragments, which are idle spaces in the schedules, can be filled as long as estimations are smaller or the same size as the fragments. By increasing inaccuracy, more fragments are

[‡]More details about the machines in Grid'5000 can be found at https://www.grid5000.fr

Table III. Overview of the node configurations for the experiments in Grid'5000. Sites are interconnected inside the same VLAN at 10Gbps.

Scheduler	Cluster	Location	CPUs' Configuration
Metascheduler	sol	Sophia	AMD Opteron 2.0 GHz
Provider 1	paradent	Rennes	Intel Xeon 2.5 Ghz
Provider 2	bordemer	Bordeaux	AMD Opteron 2.2 GHz
Provider 3	grelon	Lille	AMD Opteron 2.6 GHz
Provider 4	chicon	Nancy	Intel Xeon 1.6 GHz

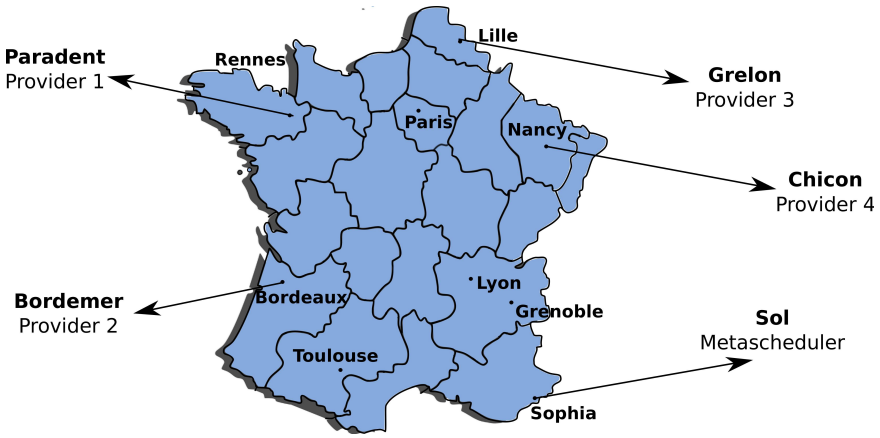
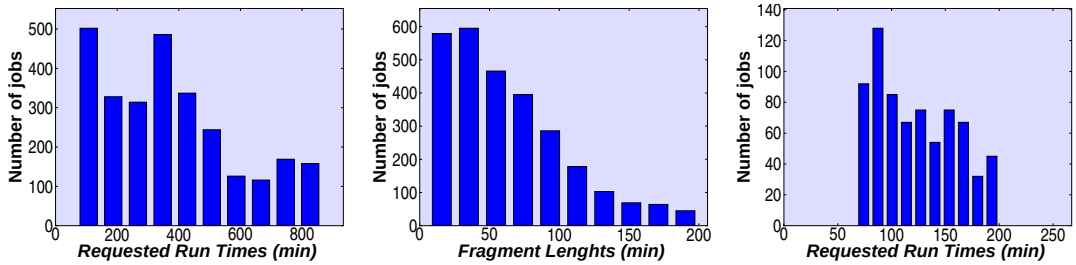


Figure 7. Location of processors in Grid'5000.

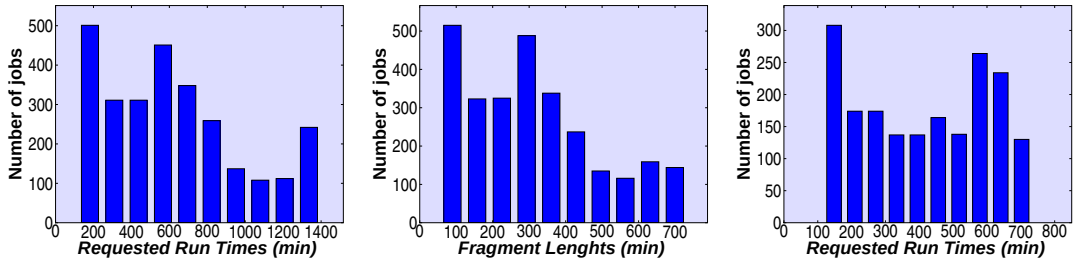
created and therefore more jobs can be backfilled. However, there is a limit in which backfilling can be explored. Figures 8 and 9 show the requested run times, fragment lengths, and number of jobs that would fit into the fragments for run time estimates with accuracy of 85% and 50%, respectively. We observe that the higher the accuracy the smaller is the number of jobs that have changes of being backfilled. For the examples of Figures 8, 9, and 10, we are not considering the submission time of the jobs. By plotting the total number of jobs that would have chances to be backfilling as a function of run time accuracy, we notice that there is a limit on the backfilling chances. Figure 10 shows that after an overestimation of 200%, the chances of backfilling become steady.

The main motivation for developing the coordinated rescheduling for BoT applications is the observation that stretch factor increases with the run time overestimations. Figure 11 presents the stretch factor for applications scheduled in multiple clusters as a function of run time overestimation for homogeneous and heterogeneous environments. Until 30% of overestimation, there is no difference between the rescheduling strategies. This happens because by this value, just a few jobs have chances of backfilling. However, after 30%, tasks of BoT applications spread over the scheduling queues due to the backfilling, thus increasing the stretch factor. The



(a) Requested run times. (b) Fragment lengths. (c) Chances to be backfilled.

Figure 8. Requested run times and fragment lengths of the workloads for accuracy of 85%.



(a) Requested run times. (b) Fragment lengths. (c) Chances to be backfilled.

Figure 9. Requested run times and fragment lengths of the workloads for accuracy of 50%.

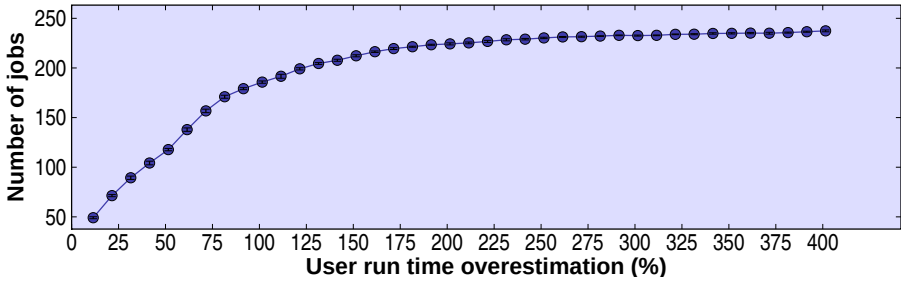
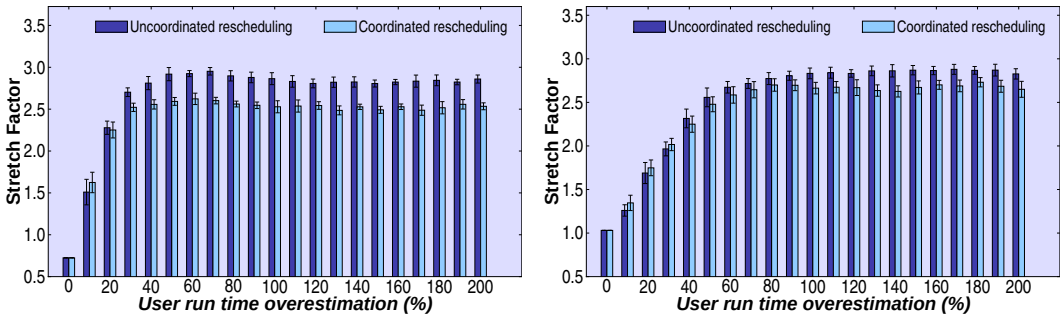


Figure 10. Backfilling limit as a function of run time overestimations.

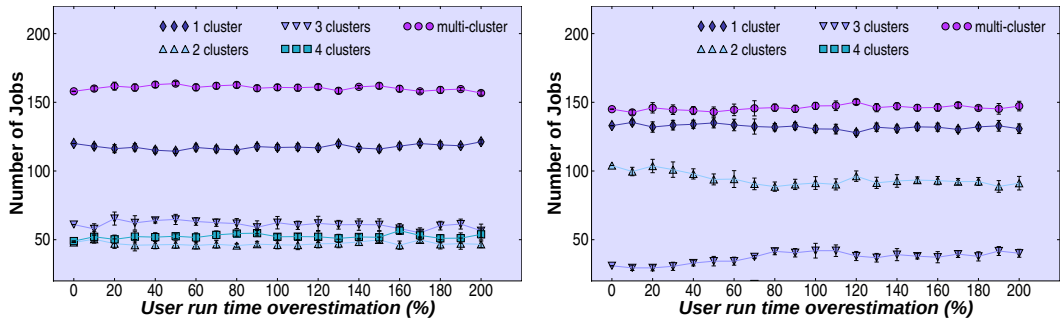
coordinated rescheduling minimizes this effect in approximately 20% and 10% for homogeneous and heterogeneous environments respectively. For the heterogeneous environment, although stretch factor is reduced using coordinated rescheduling over the uncoordinated one, this improvement is lower than in homogeneous environments (Figure 11). The reason is that applications tend to be executed in fewer clusters as the metascheduler places tasks on the

fastest cluster in order to reduce completion time). Therefore, the importance for coordinated rescheduling among providers is reduced. As showed in Figure 12, the number of clusters per job is reduced in the heterogeneous environment. Most of the applications are scheduled to one or two clusters, whereas for the homogeneous environment similar number of applications access two, three, and four clusters.



(a) Homogeneous environment. (b) Heterogeneous environment.

Figure 11. Stretch factor as a function of the run time estimation accuracy and rescheduling schema.

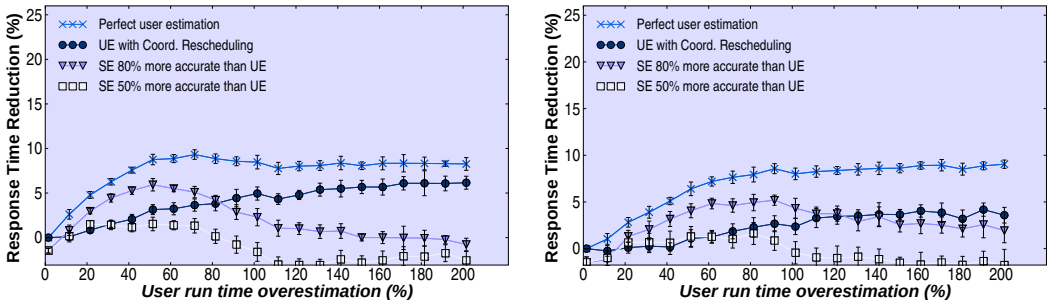


(a) Homogeneous environment. (b) Heterogeneous environment.

Figure 12. Number of clusters per job.

Reducing the stretch factor may have an impact on the user response time. Figure 13 presents the response time reduction of coordinated rescheduling and system-generated predictions in comparison to user run time estimates with uncoordinated rescheduling. We observe that the difference between the policies is higher for the homogeneous environment, since jobs are more distributed to multiple providers than in the heterogeneous environment. In addition, system-generated predictions have better improvements in the heterogeneous environment than in the homogeneous one. The reason is that incorrect work distribution in a heterogeneous

environment causes more negative effects than in a homogeneous one. We also observe that system-generated prediction policies have a similar curve shape that perfect user estimation until a certain threshold (60% for homogeneous and 70% for heterogeneous environment). After this threshold the advantage of using system-generated predictions is reduced. This happens because there is a benefit limit in backfilling (as illustrated in Figure 10) and it becomes lower than the cost paid to obtain better estimations.



(a) Homogeneous environment.

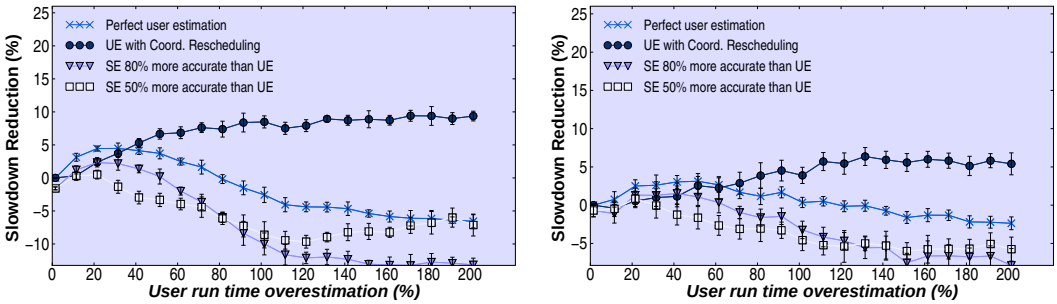
(b) Heterogeneous environment.

Figure 13. Global user response time reduction in comparison to uncoordinated rescheduling.

We also analyzed the user response time separately for multi- and single-provider jobs. Figure 15 presents the results for single-provider jobs. We observed that the increase of user overestimations actually reduces user response time for these jobs, which corroborates with previous studies on effects of run time estimates for job scheduling [33]. User response time for coordinated rescheduling produces an improvement of up to 5% in relation to uncoordinated rescheduling for these jobs. The main benefits of higher run time accuracy and coordinated rescheduling come from multi-provider jobs, as illustrated in Figure 16. It is worth noting that scheduling algorithms and techniques for reducing response time and increasing system utilization are getting more complex and every gain is important, especially if we consider a global scenario where machines are consuming a considerable amount of electricity and users are demanding higher Quality-of-Service levels.

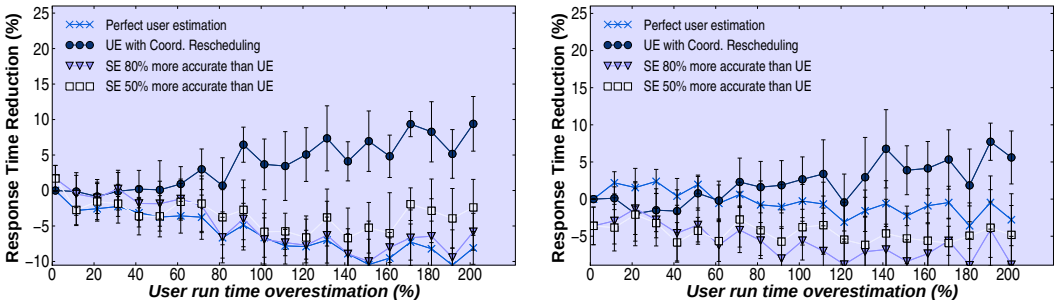
We have also analyzed the slowdown (with 10 minutes bound), which is the response time divided by the application run time. Figure 14 presents the slowdown for homogeneous and heterogeneous environments. We observe that for this metric, coordinated rescheduling presents even better results than using perfect run time estimations. This happens because this metric highlights the improvements of smaller jobs in relation to big ones—smaller jobs have more chances of backfilling than the big ones.

We have also calculated system utilization for user and system-generated estimations and uncoordinated/coordinated rescheduling algorithms. The results are similar with a difference of less than 1%. This difference may increase if we consider a competition scenario with providers offering different completion time guarantees. In this scenario, users will tend to execute their applications on providers offering shorter completion times. Figure 17 illustrates the average



(a) Homogeneous environment. (b) Heterogeneous environment.

Figure 14. Global slowdown reduction in comparison to uncoordinated rescheduling.

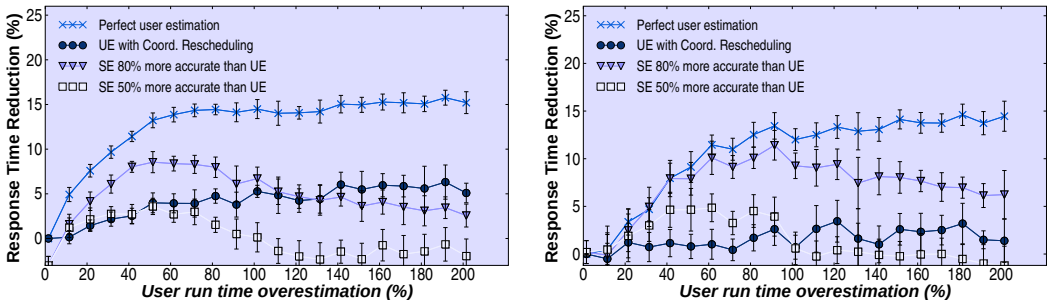


(a) Homogeneous environment. (b) Heterogeneous environment.

Figure 15. Single-cluster job response time reduction in comparison to uncoordinated rescheduling.

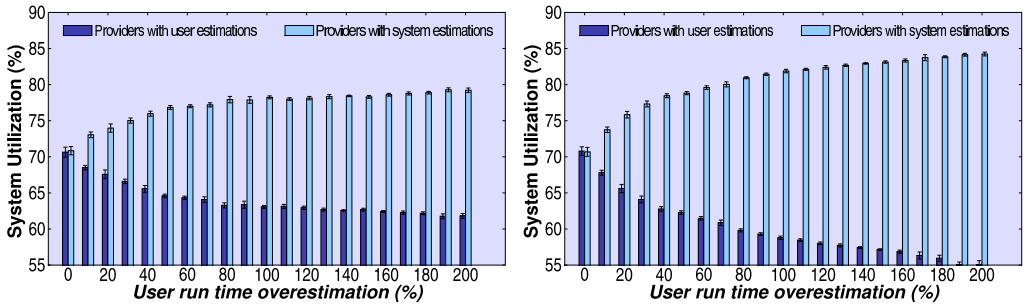
system utilization of providers with different run time estimation schemas; the higher the accuracy of run time predictions the higher the chances of attracting more users. System utilization is the percentage of the resources that have been used over time.

We have also performed experiments in Grid'5000 to evaluate possible technical difficulties to deploy the coordinated rescheduling. We selected a few workloads and compared the results from simulations and executions in the real environment. Table IV presents the scenarios and obtained results. We observed that for these experiments, both simulations and executions in the real environment provided similar results, showing the practical benefits of coordinated rescheduling. As we described in the section on coordinated rescheduling implementation, the required modification in an existing scheduling architecture is minimal. The execution time for rescheduling tasks is less than a second, since just a few bytes that specify task IDs, completion times, and IP addresses are required to be transferred over the network.



(a) Homogeneous environment. (b) Heterogeneous environment.

Figure 16. Multi-cluster job response time reduction in comparison to uncoordinated rescheduling.



(a) SE = 0.5 UE (b) SE = 0.2 UE

Figure 17. Impact of estimations on the system utilization by attracting more users through more optimized completion time guarantees.

6. CONCLUSIONS

This paper presented a coordinated rescheduling algorithm for BoT applications executing across multiple providers and the impact of run time estimates for these applications. Due to inaccurate run time estimates, initial schedules have to be updated, and therefore, when each provider reschedules tasks of a BoT application independently, other applications may not have chances of reducing their response time.

The main finding is that tasks of the same BoT can have considerably different completion times due to inaccurate run time estimates and environment heterogeneity. Coordinated rescheduling of these tasks can hence reduce user response time for both single- and multi-provider applications in approximately 5%; and slowdown reduction of up to 10%. This improvement comes from the observation that reducing the expected completion time of tasks from the same BoT independently prevents backfilling of other tasks. Moreover, in order to deploy coordinated rescheduling, metaschedulers and resource providers only require a simple

Table IV. Comparison of results from Grid'5000 and simulations.

Metric and Overestimation (%)	From simulation (%)	From real system (%)
SFactor for uncoordinated rescheduling - 50	2.92 ± 0.08	3.05
SFactor for uncoordinated rescheduling - 100	2.86 ± 0.07	2.81
SFactor for uncoordinated rescheduling - 150	2.81 ± 0.04	2.69
SFactor for coordinated rescheduling - 50	2.59 ± 0.05	2.65
SFactor for coordinated rescheduling - 100	2.53 ± 0.07	2.42
SFactor for coordinated rescheduling - 150	2.49 ± 0.04	2.56
Response time reduction - 50	3.14 ± 0.55	3.84
Response time reduction - 100	4.95 ± 0.74	6.94
Response time reduction - 150	5.68 ± 0.74	5.99
Slowdown reduction - 50	6.66 ± 0.79	6.74
Slowdown reduction - 100	8.48 ± 0.92	10.4
Slowdown reduction - 150	8.88 ± 1.29	10.07

data structure and protocol to keep track of the expected completion time of the last task of each BoT applications. System-generated predictions can serve as an alternative to coordinated rescheduling but require more effort for deployment and may not reduce user response times as much as when using coordinated rescheduling.

ACKNOWLEDGEMENTS

We thank Adam Barker and Marcos Dias de Assunção for their comments on this paper. Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr). We also thank Dr. Dror Feitelson for maintaining the Parallel Workload Archive, and all organizations and researchers who made their workload logs available. This work is partially supported by research grants from the Australian Research Council (ARC) and Australian Dept. of Innovation, Industry, Science and Research (DIISR).

REFERENCES

1. Pande VS, Baker I, Chapman J, Elmer S, Larson SM, Rhee YM, Shirts MR, Snow CD, Sorin EJ, Zagrovic B. Atomistic protein folding simulations on the submillisecond time scale using worldwide distributed computing. *Peter Kollman Memorial Issue, Biopolymers* 2003; **68**(1):91–109.
2. Smallen S, Casanova H, Berman F. Applying scheduling and tuning to on-line parallel tomography. *Proceedings of the ACM/IEEE Conference on Supercomputing (SC'01)*, ACM, 2001.
3. Altschul S, Gish W, Miller W, Myers EW, Lipman DJ. Basic local alignment search tool. *Journal of Molecular Biology* 1990; **215**(3):403–410. URL `http://dx.doi.org/10.1006/jmbi.1990.9999`.
4. Auyoung A, Grit L, Wiener J, Wilkes J. Service contracts and aggregate utility functions. *Proceedings of the 15th International Symposium on High Performance Distributed Computing (HPDC'06)*, IEEE Computer Society, 2006.

5. Netto MAS, Buyya R. Offer-based scheduling of deadline-constrained bag-of-tasks applications for utility computing systems. *Proceedings of the International Heterogeneity in Computing Workshop (HCW), in conjunction with the 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE Computer Society, 2009.
6. Lee CB, Snavely A. On the user-scheduler dialogue: Studies of user-provided runtime estimates and utility functions. *International Journal of High Performance Computing Applications* 2006; **20**(4):495–506.
7. Lee CB, Schwartzman Y, Hardy J, Snavely A. Are user runtime estimates inherently inaccurate? *Proceedings of the 10th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'04)*, *Lecture Notes in Computer Science*, vol. 3277, Springer, 2004; 253–263.
8. Tsafrir D, Etsion Y, Feitelson DG. Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Transactions on Parallel and Distributed Systems* 2007; **18**(6):789–803.
9. Nurmi D, Brevik J, Wolski R. Qbets: Queue bounds estimation from time series. *Proceeding of the 13th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, *Lecture Notes in Computer Science*, vol. 4942, Springer, 2007.
10. Smith W, Taylor VE, Foster IT. Using run-time predictions to estimate queue wait times and improve scheduler performance. *Proceeding of the International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, *Lecture Notes in Computer Science*, vol. 1659, Springer, 1999.
11. Yang LT, Ma X, Mueller F. Cross-platform performance prediction of parallel applications using partial execution. *Proceedings of the ACM/IEEE SC'05*, 2005.
12. Sanjay HA, Vadhiyar SS. Performance modeling of parallel applications for grid scheduling. *Journal of Parallel and Distributed Computing* 2008; **68**(8):1135–1145.
13. Romanazzi G, Jimack PK. Parallel performance prediction for numerical codes in a multi-cluster environment. *Proc. of the 2008 International Multiconference on Comp. Science and Information Technology (IMCSIT'08)*, 2008.
14. Jarvis SA, Spooner DP, Keung HNLC, Cao J, Saini S, Nudd GR. Performance prediction and its use in parallel and distributed computing systems. *Future Generation Computer Systems* 2006; **22**(7):745–754.
15. Sadjadi SM, Shimizu S, Figueroa J, Rangaswami R, Delgado J, Duran H, Collazo-Mojica XJ. A modeling approach for estimating execution time of long-running scientific applications. *Proceedings of the 22nd IEEE International Symposium on Parallel and Distributed Processing (IPDPS'08)*, 2008.
16. Downey AB. Predicting queue times on space-sharing parallel computers. *Proceedings of the 11th International Parallel Processing Symposium (IPPS'97)*, IEEE Computer Society, 1997; 209–218.
17. Sonmez OO, Yigitbasi N, Iosup A, Epema DHJ. Trace-based evaluation of job runtime and queue wait time predictions in grids. *Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing (HPDC'09)*, ACM, 2009; 111–120.
18. He L, Jarvis SA, Spooner DP, Chen X, Nudd GR. Dynamic scheduling of parallel jobs with QoS demands in multiclusters and grids. *Proceedings of the International Conference on Grid Computing (GRID'04)*, 2004.
19. Berman F, Wolski R, Casanova H, Cirne W, Dail H, Faerman M, Figueira SM, Hayes J, Obertelli G, Schopf JM, *et al.*. Adaptive computing on the grid using AppLeS. *IEEE Transactions on Parallel and Distributed Systems* 2003; **14**(4):369–382.
20. Jarvis SA, He L, Spooner DP, Nudd GR. The impact of predictive inaccuracies on execution scheduling. *Performance Evaluation* 2005; **60**(1-4):127–139.
21. Abramson D, Buyya R, Giddy J. A computational economy for grid computing and its implementation in the Nimrod-G resource broker. *Future Generation Computer Systems*. 2002; **18**(8):1061–1074.
22. Cirne W, da Silva DP, Costa L, Santos-Neto E, Brasileiro FV, Sauvé JP, Silva FAB, Barros CO, Silveira C. Running bag-of-tasks applications on computational grids: The MyGrid approach. *Proceedings of the 32nd International Conference on Parallel Processing (ICPP'03)*, IEEE Computer Society, 2003; 407–.
23. Iosup A, Sonmez OO, Anoep S, Epema DHJ. The performance of bags-of-tasks in large-scale distributed systems. *Proceedings of the 17th International Symposium on High-Performance Distributed Computing (HPDC'08)*, ACM, 2008; 97–108.
24. Benoit A, Marchal L, Pineau JF, Robert Y, Vivien F. Offline and online master-worker scheduling of concurrent bags-of-tasks on heterogeneous platforms. *Proceedings of the 22nd IEEE International Symposium on Parallel and Distributed Processing (IPDPS'00)*, IEEE Computer Society, 2008.
25. Kim JK, Shivle S, Siegel HJ, Maciejewski AA, Braun TD, Schneider M, Tideman S, Chitta R, Dilmaghani RB, Joshi R. Dynamically mapping tasks with priorities and multiple deadlines in a heterogeneous environment. *Journal of Parallel and Distributed Computing* 2007; **67**(2):154–169.
26. Beaumont O, Carter L, Ferrante J, Legrand A, Marchal L, Robert Y. Centralized versus distributed schedulers for bag-of-tasks applications. *IEEE Transactions on Parallel and Distributed Systems* 2008;

19(5):698–709.
27. Cirne W, Brasileiro FV, da Silva DP, Góes LFW, Voorsluys W. On the efficacy, efficiency and emergent behavior of task replication in large distributed systems. *Parallel Computing* 2007; **33**(3):213–234.
28. Haji MH, Gourlay I, Djemame K, Dew PM. A SNAP-based community resource broker using a three-phase commit protocol: A performance study. *The Computer Journal* 2005; **48**(3):333–346.
29. Lublin U, Feitelson DG. The workload on parallel supercomputers: modeling the characteristics of rigid jobs. *Journal of Parallel and Distributed Computing* 2003; **63**(11):1105–1122.
30. Macías M, Rana OF, Smith G, Guitart J, Torres J. Maximizing revenue in grid markets using an economically enhanced resource manager. *Concurrency and Computation: Practice and Experience* 2010; **22**(14):1990–2011.
31. Bruneo D, Longo F, Scarpa M, Puliafito A. Performance analysis of job dissemination techniques in grid systems. *Concurrency and Computation: Practice and Experience* 2011; **23**(11):1213–1235.
32. Shmueli E, Feitelson DG. Backfilling with lookahead to optimize the packing of parallel jobs. *Journal of Parallel Distributed Computing* 2005; **65**(9):1090–1107.
33. Tsafrir D, Feitelson DG. The dynamics of backfilling: Solving the mystery of why increased inaccuracy may help. *Proceedings of the 2006 IEEE International Symposium on Workload Characterization (IISWC'06)*, IEEE Computer Society, 2006; 131–141.